ConneXions™

The Interoperability Report

June 1990 Volume 4, No. 6

ConneXions— The Interoperability Report tracks current and emerging standards and technologies within the computer and communications industry.

In this issue:

ConneXions is published monthly by Interop, Inc., 480 San Antonio Road, Suite 100, Mountain View, CA 94040, USA. 415-941-3399. Fax: 415-949-1779.

ISSN 0894-5926

From the Editor

The *Versatile Message Transport Protocol* (VMTP) was developed at Stanford University as part of ongoing research into high performance distributed systems. VMTP provides service roughly corresponding to TCP, but since it is inherently unconnected it does not fit into the existing Internet or OSI protocol models. Instead, VMTP was designed to meet the needs of high-performance *Remote Procedure Call* systems (RPC). Tony Mason of Transarc Corporation describes the basic services provided by VMTP and compares it to existing protocols such as TCP, UDP and TP4.

A planned installment in our series *Components of OSI* is a tutorial on the X.25 protocols. I've been looking for someone to write this article for some time, but so far nobody has volunteered. When I asked Jon Crowcroft of University College London to write about X.25, he offered instead to review a book on the subject. The book review appears on page 11. Stay tuned while I continue to look for that definitive X.25 article.

If you use a hardwired "dumb" terminal to talk to your host computer system, you are used to performing certain functions from the keyboard such as freezing or interrupting output. If you talk to your host via a virtual terminal connection using *Telnet*, you expect more or less the same behavior. Unfortunately, at the time a control character is received by the operating system, there may be thousands of characters which have been sent by the remote Telnet server but not yet displayed your local terminal. This is an inherent problem with "in-band" control signals, and it is aggravated by delays introduced by wide area networks. Mitch Tasman of the University of Wisconsin-Madison discusses these problems in an article entitled "Telnet Output Discard Processing," and offers some solutions in light of the recently released Telnet LINEMODE option.

Regional networks are springing up all over the country, particularly now that the original ARPANET is being phased out. One such network is the *New England Academic and Research Network*, or NEARnet. A profile of this network appears in this issue starting on page 16.

By the way, we now have FUnet, and BARRnet, NEARnet and FARNET, it must be time to write a song or something! Send your contributions to *ConneXions*, 480 San Antonio Road, Suite 100, Mountain View, CA 94040, USA.

Finally, on page 19, a preview of next month's article on Internet routing protocols, and your chance to put your Internet site on the cover of Doug Comer's new TCP/IP book.

VMTP: A High Performance Transport Protocol

by W. Anthony Mason, Transarc Corporation

Introduction

The *Versatile Message Transport Protocol* (VMTP) is a high performance transport protocol developed at Stanford University as part of ongoing research into high performance distributed systems. Logically, it provides service roughly corresponding to TCP (e.g., reliable), but because it is inherently unconnected does not fit into the existing DARPA or OSI protocol models. Instead, it was designed to meet the needs of high-performance *Remote Procedure Call* systems (RPC). This article describes the basic services provided by VMTP and compares it to existing protocols such as TCP, UDP and TP4.

Motivation

The Distributed Systems Group at Stanford University under the direction of Professor David R. Cheriton has been actively involved in the development of the **V** distributed system which is a high-speed Interprocess Communications (IPC) based operating system. Because of the critical nature of communications to this endeavor traditional communications protocols were examined and rejected. TCP and its OSI counterpart TP4 are unacceptable in high-speed environments because of the high initial setup cost of connections and the cost of maintaining connection state; it is not unusual for a high-demand service, such as a file service, to have hundreds of active connections and such is unworkable in that environment. A traditional solution in the BSD UNIX domain has been to build a special purpose protocol using UDP. UDP, however, provides no service guarantees; mechanisms to handle reordering of packets and lost packets must be implemented by the particular protocol. What was needed was a high-performance protocol capable of providing reliable delivery but not requiring the constant maintenance. It was through this realization that VMTP was created.

Basic model

Like TCP and TP4, VMTP presumes it will be sending messages using an unreliable datagram service. From this it guarantees to its senders to provide reliable communications. In Figure 1 we provide a sample VMTP connection.

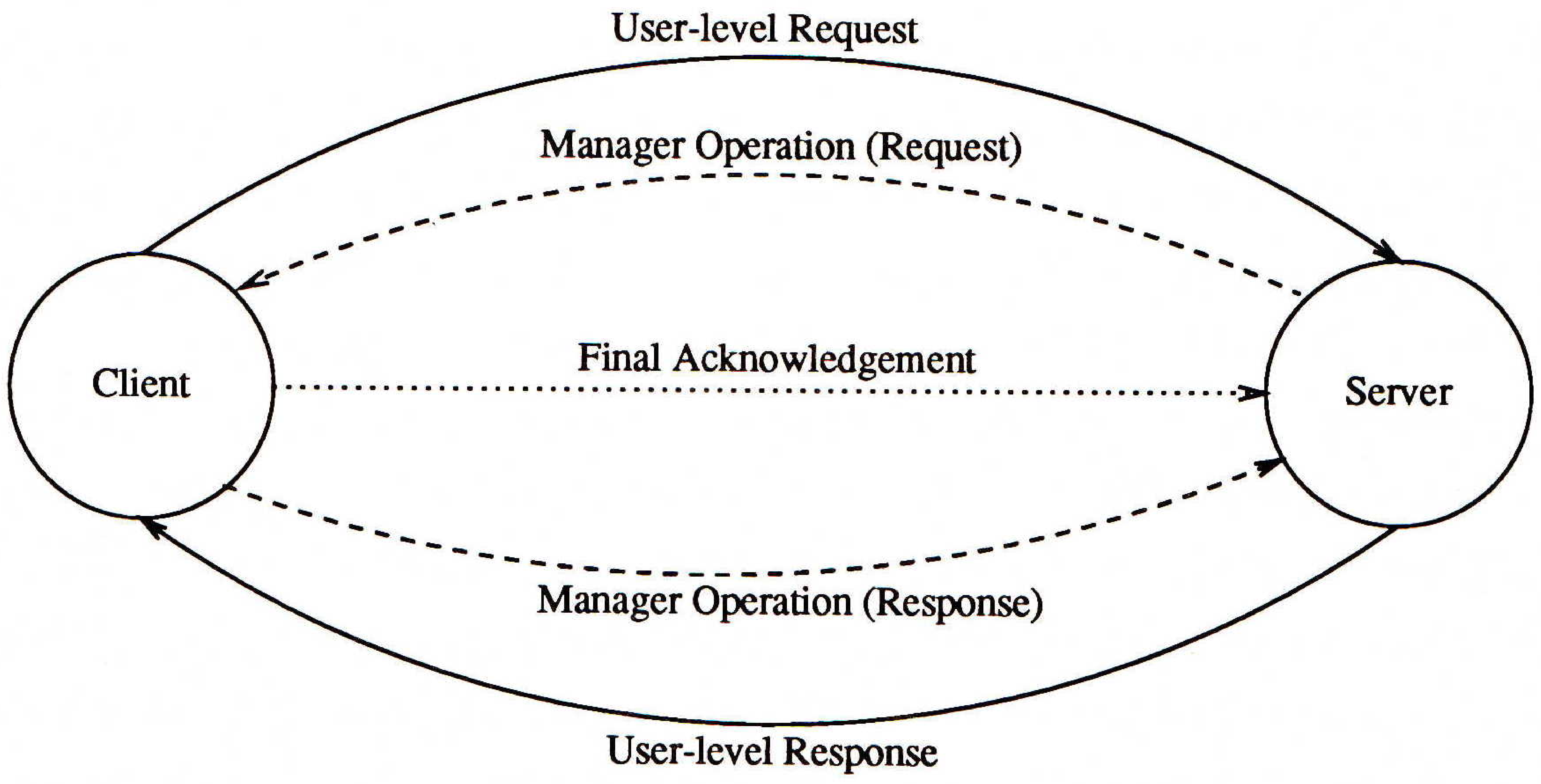

Figure 1: VMTP Communications Model

The basic VMTP model provides *transactional* communications semantics, in that a *client* communications process sends a *request* to a *server* communications process. The server *responds* to the client request. If the client wishes to continue communications, a new request will be sent. This then acts as an inherent acknowledgement of the previous communications. If the client does not wish to continue communications, an acknowledgement of the response will be sent.

VMTP requests and responses can be arbitrary sized messages. VMTP handles any problems associated with retransmission, reordering, and data verification. Neither the client nor the server need handle such error conditions.

Independent of the client/server communications, the VMTP modules also communicate between themselves. Each VMTP implementation has a pseudo-client known as a *manager*. For instance, the first time a message is received from a client, the server's VMTP manager might (depending upon the implementation and security guarantees requested by the server) *call back* to the client's VMTP manager. These managers will then exchange sufficient information to carry out communications. This method is quite different than either the traditional unconnected case, where each message contains sufficient information to provide "connection state," or connected case, where communications is preceded and followed by an exchange of such state information.

Because of this ability to discard state, since it may be requested if needed, it is possible to implement VMTP in constrained conditions (e.g. within a bootstrap loader where maintaining much state may be inconvenient.) Further, one of the difficulties apparent when using TCP as a connection base for a heavily used server machine is the cost of maintaining state. With VMTP, it is possible to consider the saved state as a *cache*. Thus, we can obtain exactly the type of semantics desirable in such high-load situations—those clients using the system most heavily will always have their information cached without prohibiting lighter using clients access.

Idempotent response

In addition to this basic model, VMTP provides a rich set of options which allow more specialized applications full control of the message, if desired; if none are specified VMTP provides those fundamental services described above. For instance, a server may specify that a response is *idempotent*, that is the data should *not* be retransmitted. This is precisely the service required by a time server; the requester wants a definite time but it makes little sense to retransmit the time as it *was* if it is lost in the communications system. A client may specify that a request is a *datagram*, that is no response is expected. This is useful for real-time applications such as data acquisition where each individual point of data is of little significance in itself but delivery should be accomplished with high probability and thus *most* of the data will be delivered.

Latency versus throughput

Finally, measurements of typical traffic patterns on local area networks reveals that most messages are either very small (typically the minimum allowed by the system) or are very large. For instance in the file server example the read request would be small while the response quite large. For small messages the goal is to provide high-speed communications, keeping overall latency low.

continued on next page

VMTP *(continued)*

For large messages the goal is to provide large data paths, keeping overall throughput high. Traditionally this has been seen as a trade-off but VMTP attempts to obtain both low latency and high throughput. Later I will describe the Stanford implementation which indicates both *can* be accomplished.

Features

A primary strength VMTP offers is its ability to satisfy the needs of a large number of communications users; much of this is accomplished by providing a collection of services to compliment the basic underlying reliable communications model described in the previous section.

Asynchronous communications

A traditional RPC protocol enforces the pure procedural model upon the programmer in that each call blocks until completion. VMTP provides a mechanism for the programmer to request asynchronous communications service from the VMTP communications layer. This allows the programmer to write programs which do not block and can hence perform some multiprogramming. For example, a domain name server could send a request to the higher level domain name server while continuing to process incoming naming requests and still have reliable communications semantics not available with UDP.

Streaming

Providing support for an arbitrary sized data segment is challenging. Many RPC protocols provide some cap upon the size of the possible data blocks they will transmit. Typically, an applications program must then cycle through its data pool sending the data in several large chunks—in effect creating an extension to the protocol. To alleviate this problem on the part of the programmer, VMTP uses *streaming* of data which allows the sending of an arbitrary number of data blocks.

In addition to merely being able to send arbitrary sized data blocks, it also allows the applications programmer to send only a *portion* of the entire data segment. This can be useful if there are limitations upon the total available memory in the server machine. For example, if a client asks the server to send a file larger than the server's available virtual memory, the server can stream the file by sending individual smaller pieces of the file. VMTP will provided the file in pieces to the client or even as a single 40MB file, as requested by the client.

Groups

The group communications support of VMTP seems to fulfill two primary needs: location and replication. For example, location of a particular service is quickly located by sending to a *group* of such servers; naming is one excellent example where such location capabilities are useful. An example of replication is for program binaries. In this case, the file servers could each maintain a copy of the requisite binaries. By sending to the group of such servers the appropriate binary can be obtained and the failure of a single server often does not impact the user.

Authentication

Because of the needs of some applications for authentication, VMTP provides an interface into such authentication. Although the RFC itself provides little direction into the *types* of such authentication, those provisions necessary are present (e.g., the *callback* mechanism between VMTP managers.)

In the future, as VMTP evolves, it is anticipated that additional specifications of newer *authentication domains* will emerge.

Security

VMTP attempts to provide *reliable* communications between communicating entities. In designing VMTP, we felt it necessary to provide such reliability in the face of active attempts to either read or modify the actual messages. In order to prevent this, we introduced facilities for *encryption*. Each authentication domain defines what encryption mechanisms will be available within it.

The RFC defines a single authentication domain and, within that authentication domain two possible types of encryption: none and CBC DES. Any VMTP implementation which *does* provide authentication services will insure that communications between the two entities are secure (additionally, there are also guarantees that communications between the VMTP managers will be protected.)

Entity Identifiers

VMTP provides an addressing scheme which is independent of the particular underlying protocol. A VMTP communications entity is identified by an *entity identifier*, such as is shown in Figure 2 The RAE bit is used to identify a *Remote Alias Entity* which means that the address is actually an *alias* for a process outside this addressing domain. The LEE bit identifies that the entity which transmitted the packet used a little-endian byte order; if it is not set, a big-endian byte order is assumed.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R| |L|R|                                                       |
|A|0|E|E|       Domain-specific structure                       |
|E| |E|S|                                                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                Domain-specific structure                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: VMTP Entity ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R| |U|R|                                                       |
|A|1|G|E|       Domain-specific structure                       |
|E| |P|S|                                                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                Domain-specific structure                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: VMTP Group Address

Each VMTP *addressing domain* defines the particular structure of the VMTP entity identifier. For example, in addressing domain 1, used in IP networks, the first four bits are used as the flags fields, the next 28 are used as a timestamp and the final 32 bits are an IP address of the machine upon which the entity was created. In addressing domain 3, used by the **V** distributed system, the last 60 bits are dynamically allocated by *probing* for blocks of available addresses.

continued on next page

VMTP *(continued)*

Group Identifiers

In addition to the entity identifiers, there are corresponding *group identifiers*. The format of group addresses is shown in Figure 3. Each VMTP group address is also defined by its *addressing domain*. The RAE bit retains its meaning as before. The UGP bit is used to indicate if the group is *unrestricted*. A *restricted* group is one in which some authority exercises control over membership, while an *unrestricted* group is one which any entity may join.

Each domain must guarantee that addresses are *t-stable*, that is that they are not re-used frequently (within some time *t*.) Each domain is free to implement its own mechanism for such stability (and, in fact, the mechanisms for the two domains described in the RFC are quite different.) The intention here is to guarantee that when an address is re-used any previous references to that address will no longer exist.

Coresident addressing

VMTP relies upon the concept of *coresident addressing* heavily. For example, a VMTP manager locates the VMTP manager for a particular entity by sending to the manager *coresident* with the entity. Thus, the concept of a *coresident process* is any process which shares the *same* VMTP implementation; this typically overlaps with processes sharing the same *machine*.

Because of the existence of both coresident addressing and group addressing, it is possible to provide flexibility in locating actual services. Unlike normal addressing, however, the *message* is delivered as if it were addressed to the coresident process. Only when it reaches that final destination does it actually resolve to the intended recipient. Thus, it is possible to *not* know the particular name of an entity's VMTP manager, but rather to send to the *VMTP Manager's Group* coresident with the entity itself.

Implementation

VMTP has been implemented for both BSD UNIX as well as the **V** system. The BSD implementation relies upon the presence of *IP Multicasting*. In this section we will describe the BSD UNIX implementation.

Message processing

Each VMTP message consists of a fixed sized header, a variable data section and a fixed size trailer. In Figure 4 we demonstrate the basic request packet format.

The *client*, or initiator of a request, has a variety of options when sending the message initially:

- *Normal Request:* the request is sent and the client process awaits the response from the server.
- *Datagram Request:* the request is sent and the client process does *not* await the response from the server. In this case, no response is expected at all.
- *Asynchronous Request:* the request is sent and the client process does not block awaiting the response from the server. Instead, the client later requests the response (or responses) from its manager.
- *Multiple Responses Requested:* the request is sent to a group and the client expects to receive more than one response. The client requests the response(s) from its manager.

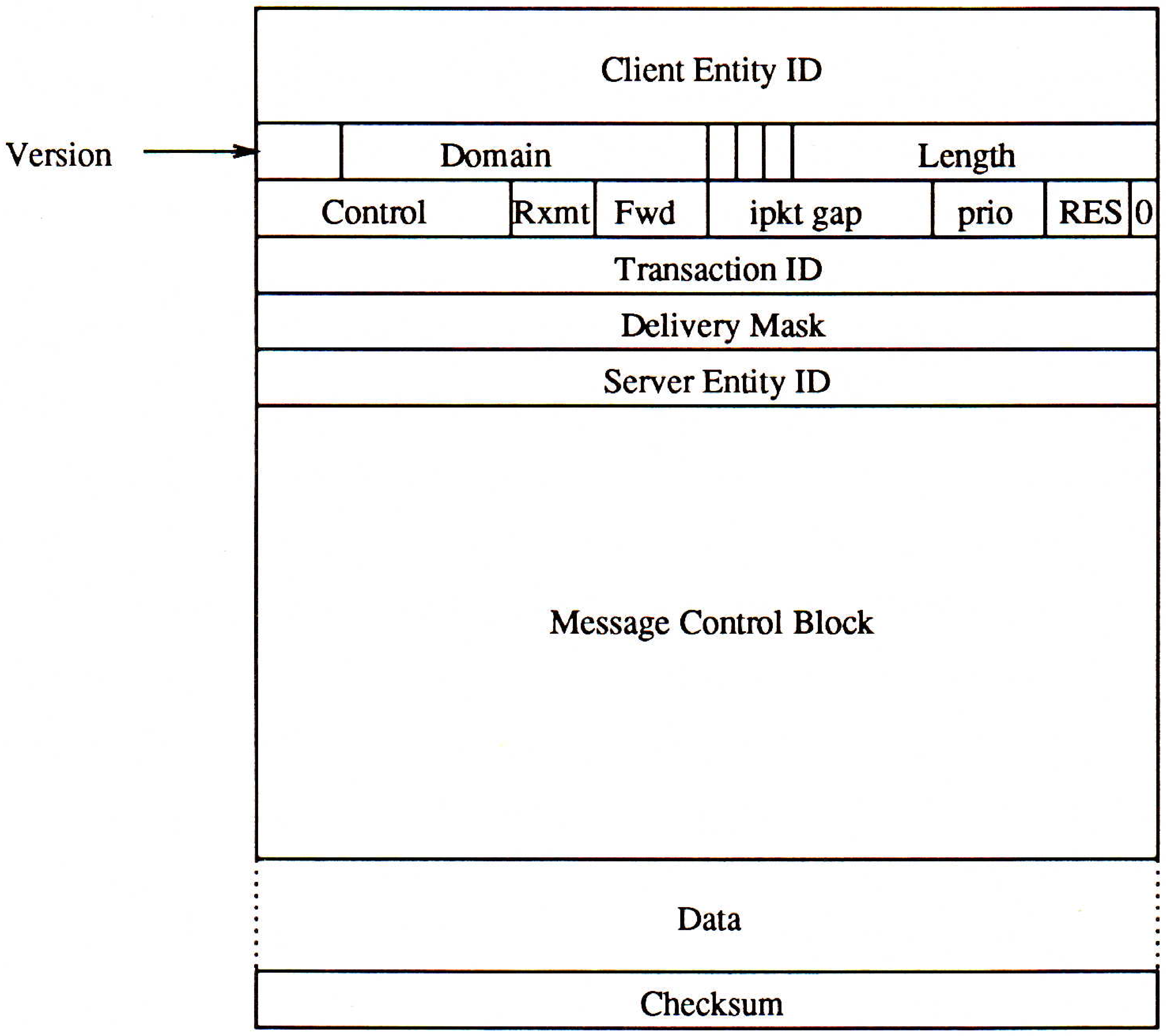

Figure 4: VMTP Request Packet

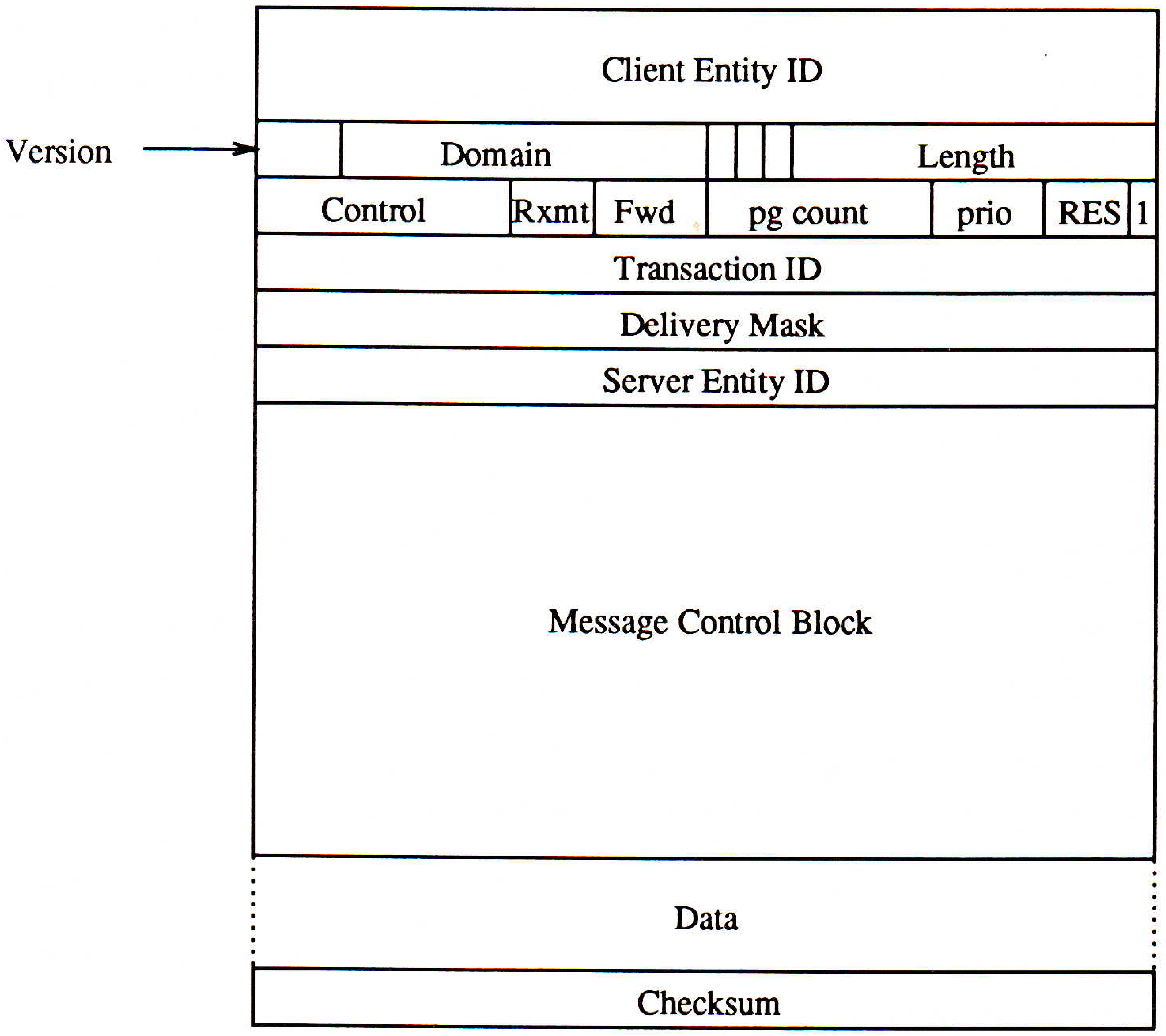

Figure 5: VMTP Response Packet

continued on next page

VMTP *(continued)*

The request packet begins with the client entity identifier, where the client is the originator of the message. Upon receipt of such a message, we check to determine if we have any information about this client cached. If so, we further check to see if this message is a continuation of a partially received message or a new message. We collect all such partial messages until the entire message has, in fact, been received from the remote system.

Once the entire message has been received and reassembled, it is queued on the message queue for the intended recipient. If the process in question is blocked waiting for a request, it is unblocked and rescheduled. If the process is *not* blocked waiting for a request, when it calls to receive a new request, it will immediately be handed the first queued request. In the BSD implementation the number of queued messages is set by the actual server process itself (and is subject to an overall system limit.) An attempt to queue more messages results in the server's VMTP manager sending a BUSY message to the client's VMTP manager.

Forming the response

A server processing a request has a variety of options. Notably, the server can:

- *Drop the request:* A server may simply ignore a request by sending a DISCARD message as the response to any message. For example, this happens for group requests, where if a server cannot answer, he is merely to ignore the request.
- *Forward the request:* A server may send the message to a *different* server for processing. For instance, a request might initially be sent to a name service who performs the appropriate location and then forwards the message to the actual service provider. The VMTP supporting version of Sun's *portmapper* uses the forwarding mechanism.
- *Respond:* A server may simply choose to answer the request.
- *Respond idempotently:* A server may wish to send a message but *not* have the response retransmitted. The canonical example of this is a *time service* which should not retransmit if the original response is not received. Instead, this means if the request is received again, it should be requeued for the server.
- *Postpone response:* It is not necessary for a server to immediately respond to a request. It may be necessary to perform additional work prior to the actual response. If that work can be interleaved with the handling of other requests the server is free to continue obtaining new requests prior to responding to older ones. Such requests are maintained on the request queue (and hence count against the quota) until a response is sent.

The BSD implementation does not presently support arbitrary sized messages, although significant work in that direction had been made. In our use of VMTP at Stanford, we had ported two different applications: the game *Mazewar* for the X Window System, and Sun's RPC package (Version 3.9), including adding support to *rpcgen* to provide support for VMTP. Both provide detailed examples of how VMTP may be used in existing applications.

For *Mazewar* the primary use is as a group management facility. *Mazewar* is a totally distributed game (no centralized server) which previously used replicated UDP datagrams. With VMTP, *Mazewar* players can create games on the fly, list existing games, and play across the Internet. Most noticeable was that, because of efficient use of the underlying network capabilities, the game played well even with several players on different networks.

For Sun's *RPC* the primary use was to demonstrate that existing applications could be retrofitted to support VMTP. After modifying the stub-compiler *rpcgen* to provide VMTP support, the RPC name service (*portmapper*) was also modified to support VMTP using applications as well, including the forwarding of requests to the actual service provider. With the sample programs in the package, it was possible to quickly add support for VMTP without changing the actual program.

In both cases, we utilized VMTP for applications ill-suited to the older semantics provided by UDP (as in the *Mazewar* case) or TCP (as in the case of the RPC example programs.)

Performance

As part of the VMTP development effort we emphasized providing detailed quantitative evidence that VMTP could be not only flexible, but also perform well. The evaluation of VMTP performance was done using SPIMS, a measurement tool developed by the Swedish Institute of Computer Science. The results we obtained are summarized in Figure 6.

Protocol	Response time (milliseconds)	Query time (milliseconds)	Maximum throughput (KBytes/sec)
VMTP	6.0	8.6	307
TCP	6.9	22.7	282
SunRPC	9.2	15.3	235
UDP	6.4	8.6	326

Figure 6: VMTP performance information

These results were derived from measurements between two Sun-3 workstations, one a Sun-3/50 and the second a Sun-3/180. More details regarding the results reported here are discussed in [3].

Future directions

Since the time the RFC was published in February, 1988, several questions have been raised, and some changes have been proposed to the actual protocol. Notably:

- *Addition of a timestamp:* In order to provide better control over packets within the network, each trailer will consist of both a *timestamp* and a *checksum*.

- *Division of transmission and control information:* Much of the information in a VMTP message deals with issues of *controlling* how the message is to be handled. Independent of this control information is transmission information, used in assembling and verifying the actual contents of the entire message have been received. After discussion, it was decided that the *control* information was not used until after the packet was received, hence its inclusion in each packet was extraneous.

continued on next page

VMTP *(continued)*

- *Refinement of the asynchronous/stream interface:* The asynchronous/stream interface was modified. It would consist of a series of interface calls: *StartRequest, ContinueRequest, EndRequest*, and *GetResponse* for the client, and *StartResponse, ContinueResponse*, and *EndResponse* for the server.

- *Independent description of control protocol:* Once the control information is separated from the transmission information, it seemed desirable to allow multiple implementations of a control protocol. The default then would be to describe a standard control protocol which would be used by VMTP managers and most applications programs, but also allow specialized applications the ability to define their own control protocol. This standard control protocol appears much as the *message control block* did in the RFC version of VMTP.

- *Forwarding support modified:* The manner in which forwarding is accomplished would change. It would use a nested-call model with sufficient information in the control block to allow "short-circuit" responses to the original client.

- *No fixed data block size:* To provide additional flexibility, each individual packet may now contain different byte counts. This provides support for newer work in MTU-discovery, where it may be desirable to adjust this amount on the fly. (The MTU is the *Maximum Transmission Unit*, or the largest IP datagram allowed on a particular physical network, e.g., an Ethernet.)

These changes are reasonable attempts to provide VMTP with an excellent range of functionality without complicating the protocol unnecessarily. Indeed, the changes, as described make VMTP *more* decomposable for implementors.

A mailing list exists for the discussion of VMTP topics. To join, send a message to `vmtp-ip-request@gregorio.stanford.edu`. VMTP code for 4BSD systems is publicly available via anonymous FTP from host `gregorio.stanford.edu`.

References

[1] Cheriton, D. R., "VMTP: The Versatile Message Transport Protocol," RFC 1045, February 1988.

[2] Cheriton, D. R. (Private Communications), "Proposed Changes to VMTP," May 12, 1989.

[3] Nordmark, Erik, and D. R. Cheriton, "Experiences from VMTP: How to achieve low response time," in *Protocols for High-Speed Networks*, edited by Harry Rudin and Robin Williamson, North-Holland, 1989, pp. 43–54.

[4] Nordmark, Erik, and P. Gunningberg, "SPIMS: a tool for protocol implementation performance measurements," in *Proceedings of 13th Conference on Local Computer Networks*, IEEE, Minneapolis, MN, October 1988.

W. ANTHONY MASON received his B.S. (1987) in Mathematics from the University of Chicago. From 1987 to 1989 he worked with the Distributed Systems Group at Stanford University under Professor David Cheriton, working with the development of VMTP. He currently works for Transarc Corporation, a company specializing in distributed systems software for UNIX platforms.

Book Review

X.25 Explained, 2nd edition, by R. J. Deasington. First published by Ellis Horwood Books, in the *Computing Science* Series in 1988. ISBN 0-7458-0110-2, 131 pages.

Audience

The inside of the front jacket contains a description of the intended audience: Computer Scientists, mainly network and system implementors.

Organisation

This is an excellent slim volume which is aimed squarely at the practical market. The book is divided into 6 chapters:

1. Introduction to the ISO 7 layer model
2. Physical Level
3. Link Level
4. Network Level
5. Transport Level
6. Triple X

Chapter 1 describes the seven layer model and is as good a brief overview as anywhere.

Chapters 2, 3 and 4 contain the bulk of the useful material, which consists of a compact and readable step-by-step explanation of the X.21 line interface, the link level frame structure and protocol, and the X.25 Packet Level Protocol.

Each level is explained with short examples of the behaviour of the protocol for call setup, tear down and normal data transfer.

Some explanation of the extended format packet types for more complex operation of PLP is included.

Some of the material includes comments on example defaults from one or two known PTT implementations.

Chapter 5 on the transport protocol has less practical value outside the UK (since the more pragmatic discussion is centered around the "Yellow Book" transport service, and the ECMA Transport).

For anyone who has ever struggled with PAD access to remote computing facilities, the final chapter on Triple X (X.3, X.28 and X.29, the terminal access family of CCITT protocols) is very useful. A list of parameters and their use is included, and how the protocols map onto the virtual circuit is explained.

A short bibliography covers the relevant standards documents.

Recommended

If you have to work with X.25, read this book first! —*Jon Crowcroft*

IETF Meeting Dates and Sites

The Internet Engineering Task Force (IETF) will meet in plenary session at the following locations:

July 31 – August 3, 1990	University of British Columbia
December 4 – 7, 1990	NCAR /University of Colorado-Boulder
March 1991 (tentative)	Washington University (St. Louis)

—*Greg Vaudreuil*

Telnet Output Discard Processing

by Mitchell Tasman, University of Wisconsin-Madison

Introduction

Most operating systems provide a character (or string of characters) which, when typed from a hardwired terminal, will cause pending output to be discarded. Some operating systems provide the ability to explicitly *Abort Output*; in others, pending output might be discarded as part of an *Interrupt Process* function. The command characters assigned to these functions might be fixed, or might be configurable by the user, depending on the operating system.

A user on a terminal connected via the *Telnet* protocol [1, 6] will expect behavior similar to that experienced on a hardwired terminal. Unfortunately, at the time an Abort Output command character is received by the operating system, there may be thousands of characters which have been sent by the Telnet server but not yet displayed on the user's terminal. These characters might be buffered at the source or destination transport layers, or in transit within the network. Also, in the case of a wide-area network, it is likely that there was a significant delay between the time the user typed the command character and the time that character was received by the operating system.

A user on a low-speed terminal may find it unacceptable to wait while the pending output is displayed. This article provides design guidelines for the Telnet client and server, as well the TCP implementations, in order to minimize the delay before output ceases.

The problem

Ideally, the Telnet client would be able to begin to discard output as soon as the user types one of the command characters which causes the remote operating system to discard pending output. Unfortunately, until the definition of the Telnet LINEMODE option [2, 3], it was not possible for the Telnet client to be aware of the OS-specific command characters which are assigned to the Abort Output and Interrupt Process functions. Thus, unless the LINEMODE option has been implemented, the only possibility is to pass the command characters transparently through to the remote operating system, and have that operating system initiate any appropriate action.

This article will first discuss Telnet output discard processing without the LINEMODE option. Then, the advantages afforded by the LINEMODE option will be addressed.

Discard mechanism

When an operating system receives a command character on a hardwired terminal which causes output to be discarded, the operating system will send a "discard" signal to the terminal driver. Upon receipt of the discard signal, the terminal driver will discard the contents of its output buffer.

The job of the Telnet server is to, upon receipt of a discard signal, emulate the action of a hardwired terminal driver as closely as possible. Herein lies the first caveat:

Caveat

The Telnet data stream contains embedded commands (including option negotiations). Output from the Telnet server cannot be discarded indiscriminately, or these commands may be lost. Thus, even during a discard, the Telnet client must scan the incoming stream for embedded commands; only normal characters can be discarded.

Upon receipt of a discard signal from the operating system, the Telnet server must as quickly as possible notify the Telnet client that a discard is in progress. This implies two things:

- The notification should be attached to the very next TCP segment sent to the client, even if there are several thousand bytes of TCP data buffered at the server's machine.

- As soon as a TCP segment with a discard indication arrives at the client's machine, the client should be signaled. This is true *even if* the TCP segment arrives out of sequence or is a retransmission. Note that the data contained in the segment should *not* be presented out-of-sequence. The client simply needs to know to begin discarding.

Not surprisingly, this requires careful coding of the TCP implementations used by both the server and client; this issue will be addressed again later in the article.

TCP Urgent

So, how is the discard notification accomplished? If the "Urgent" bit is set in a TCP segment, the segment contains the identification of an "interesting" byte in the data stream. The sequence number of the "interesting" byte is calculated as follows:

sequence number = Segment Sequence Number + Urgent Pointer

Warning!

Certain pages of the TCP Specification (both RFC 793 [4] and MIL-STD 1778 [5]) state that this calculation yields the sequence number of the byte *following* the "interesting" byte. This is *wrong!* Section 4.2.2.4 of "Requirements for Internet Hosts—Communications Layers" [7] provides a clarification. Please check your TCP implementation.

Telnet Synch

Whenever the Telnet server receives a discard signal, it inserts the two-character sequence IAC DM in the outgoing data stream, and instructs its TCP to mark the second character (DM) as "interesting." This sequence is known as the Telnet "synch" signal. Note that multiple discards are handled without difficulty; outgoing TCP segments will always identify the latest (i.e., highest-sequence-numbered) "interesting" byte.

Action by the Telnet client is fairly straightforward. When an "interesting" byte identification arrives at the Telnet client's machine, the client will be in one of two states:

1 *Processing incoming data normally*: The TCP implementation will immediately notify the client that there is an "interesting" byte somewhere ahead in the data stream, and the client will enter State 2.

2 *Processing a discard* (handling embedded commands, but discarding normal data) until the "interesting" byte has been read and an IAC DM sequence has been found: The TCP implementation will simply update its record of the "interesting" byte's sequence number. Thus two discards in quick succession will be handled as a single longer discard: the client will find the first IAC DM, but discover that the "interesting" byte hasn't yet been read.

continued on next page

Telnet Output Discard Processing *(continued)*

Note that the preceding discussion assumes that:

- An "interesting" byte is read in-sequence. If byte 101 is "interesting," it will be read after bytes 99 and 100. For Telnet, it's useless to be able to read the "interesting" byte out-of-sequence. Instead, the TCP implementation informs the Telnet client asynchronously that an "interesting" byte lies somewhere ahead in the data stream.

- After a TCP read, the Telnet client is able to inquire whether the "interesting" byte has been read. This is necessary so that the client can determine when to stop discarding.

Hint

Because of the specification ambiguity regarding the calculation of the sequence number of the "interesting" byte, some extant TCP implementations may incorrectly mark the byte *following* the DM as "interesting." A strict interpretation of the Telnet specification will result in the client discarding output forever, because the "interesting" byte will not yet have been read at the time the IAC DM is found, and the client will search forever for another IAC DM sequence. Thus, it is safer to unconditionally terminate the discard after the "interesting" byte has been read, even if an IAC DM has not been located. Your users will be much happier!

TCP details

It was previously mentioned that careful coding of the server and client TCP implementations is necessary for optimal performance. In particular, every TCP segment sent by the server's machine should contain the latest "interesting" sequence number. If possible, this should even apply to retransmissions. The client machine's TCP should immediately check all incoming segments for updated "interesting" byte information. The goal in both cases is to ensure that the Telnet client is notified of the discard absolutely as soon as possible. These implementation guidelines become more critical as the TCP receive window size of the client is increased; the amount of TCP data buffered in the server and client machines can become very large indeed!

LINEMODE

The LINEMODE option significantly enhances the functionality of the Telnet protocol. Among its many features, the LINEMODE option gives the Telnet client the ability to recognize the OS-specific command characters assigned to the Abort Output and Interrupt Process functions.

The Telnet client is then able to translate the OS-specific command characters into the standard representations defined by the Telnet protocol: IAC AO for Abort Output and IAC IP for Interrupt Process. After sending an IAC IP or IAC AO, the Telnet client sends a "synch" signal, which causes the Telnet server to discard pending input (but *not* the IAC IP or IAC AO).

When the Telnet server receives an IAC AO, it *must* send a "synch" signal to the Telnet client, in order to discard pending output. Upon receipt of IAC IP, the server *may* send a "synch" signal. See section 3.2.4 of "Requirements for Internet Hosts—Application and Support" [8] for more details concerning required and optional uses of IAC IP, IAC AO, and "synch."

An optimization

The Telnet client can employ an optimization in order to more quickly discard output. When the client sends a command (such as IAC AO) which will result in pending output being discarded, the client can immediately start discarding output. The problem is to decide to *resume* displaying output.

The recommended method [8] is for the client to send a DO TIMING-MARK option negotiation immediately after the (command, "synch") sequence, and to discard output until a WILL TIMING-MARK or WONT TIMING-MARK reply is received from the Telnet server.

Footnote for users of 4.3BSD UNIX & derived works

The TCP implementation provided with an early version of the Berkeley UNIX enhancements contained the "off by one" Urgent Pointer calculation error. When an application requested that a byte be marked as "interesting," the kernel would instead mark the *following* byte. When the kernel received an "interesting" byte, the application would be informed that the *previous* byte was "interesting."

Rather than change the behavior of the TCP implementation, Berkeley chose instead to compensate in the applications. For example, when the latest version of Telnet server sends a "synch" signal, it tells the kernel that the IAC character should be marked as "interesting." The kernel (incorrectly) marks the following character (the DM) as "interesting," and this yields the correct result.

References

[1] Postel, J. & Reynolds, J., "Telnet Protocol Specification," RFC 854.

[2] Borman, D., Editor, "Telnet Linemode Option," RFC 1116.

[3] Borman, D., "The Telnet LINEMODE Option," *ConneXions*, Volume 4, No. 5, May 1990.

[4] Postel, J. B., "Transmission Control Protocol," RFC 793.

[5] Feinler, E. J., Jacobsen, O. J., Stahl, M. K., & Ward, C. A. (Editors): *DDN Protocol Handbook,* pp 1-147–1-324, "Military Standard Transmission Control Protocol," MIL-STD-1778.

[6] Shein, B., "The Telnet Protocol," *ConneXions*, Volume 3, No. 10, October 1989.

[7] R. Braden (Ed.), "Requirements for Internet Hosts—Communication Layers," RFC 1122.

[8] R. Braden (Ed.), "Requirements for Internet Hosts—Application and Support," RFC 1123.

MITCHELL TASMAN is currently a Research Assistant at the UW-Madison Computer Sciences Department, where he is pursuing a Ph.D. He previously worked as a systems engineer for BBN Communications Corporation. His projects at BBN included a reasonably successful effort to rationalize output discard processing in the various BBN implementations of Telnet over TCP/IP. Mr. Tasman received an M.S. in Computer Sciences from UW-Madison in 1988, and an S.B. from MIT in 1983. He can be reached via e-mail as `tasman@cs.wisc.edu`.

Profile: NEARnet

by NEARnet staff

Introduction

During the mid-nineteen eighties it became apparent that New England needed a high-speed data communications network to serve its academic and research communities. When DARPA announced plans to dismantle the ARPANET in February 1988, northern New England accounted for 71 of its 258 host connections. NEARnet, the *New England Academic and Research Network*, was founded in mid-1988 by the chief information officers of Boston University, Harvard University, and the Massachusetts Institute of Technology to replace New England's defunct ARPANET links. Today, NEARnet's 40 members include universities, high-technology companies, and non-profit organizations.

Funding

Most of NEARnet's initial funding came from private funds provided by the member organizations themselves.The remainder was provided by DARPA, including funding for a portion of the capital equipment and modest support for network operations in each of NEARnet's first two fiscal years. However, the NEARnet business model calls for complete financial self-sufficiency by the end of 1990.

Organization

Under a written agreement between the founding universities, MIT is the contracting agent for NEARnet. In December 1988, BBN Systems and Technologies Corporation (BBN STC), a subsidiary of Bolt Beranek and Newman Inc., was awarded a contract to serve as the facility operator of NEARnet. NEARnet is managed by a Steering Committee with representatives from the founding universities and BBN STC. A Planning Committee composed of seventeen representatives from member and potential member organizations meets quarterly to review progress and advise the Steering Committee on policy and network activities.

Network services

NEARnet's top priority is to provide its members with reliable connections to the Internet. BBN STC technical staff monitor the status of the network, respond to failures, upgrade and reconfigure software and hardware, stock spare parts, and install new sites. NEARnet staff work closely with providers of gateway hardware, communications circuits, and microwave equipment to resolve problems with a minimum of downtime. They also maintain a 24-hour hotline for NEARnet members who have urgent technical problems.

One of the keys to providing reliable network service is the ability to manage operations problems. NEARnet staff help members solve a variety of networking problems, whether the problem originated on NEARnet or not. When a user reports an operational problem or when operations staff notice an outage, an online "trouble ticket" is created using a relational database that contains complete information about each member site. All staff members have access to trouble tickets and use them to log the steps they take to resolve each problem. NEARnet members can get information on trouble tickets by calling the hotline or by subscribing to the `nearnet-outages` mailing list.

In addition to providing network operations services, NEARnet offers a variety of user services. NEARnet holds semi-annual technical and user seminars to acquaint both novice and experienced network users with NEARnet. The seminars are open to NEARnet members and non-members. Topics covered at past seminars include routing in the Internet, using the Domain Name System, and information resources available in the Internet.

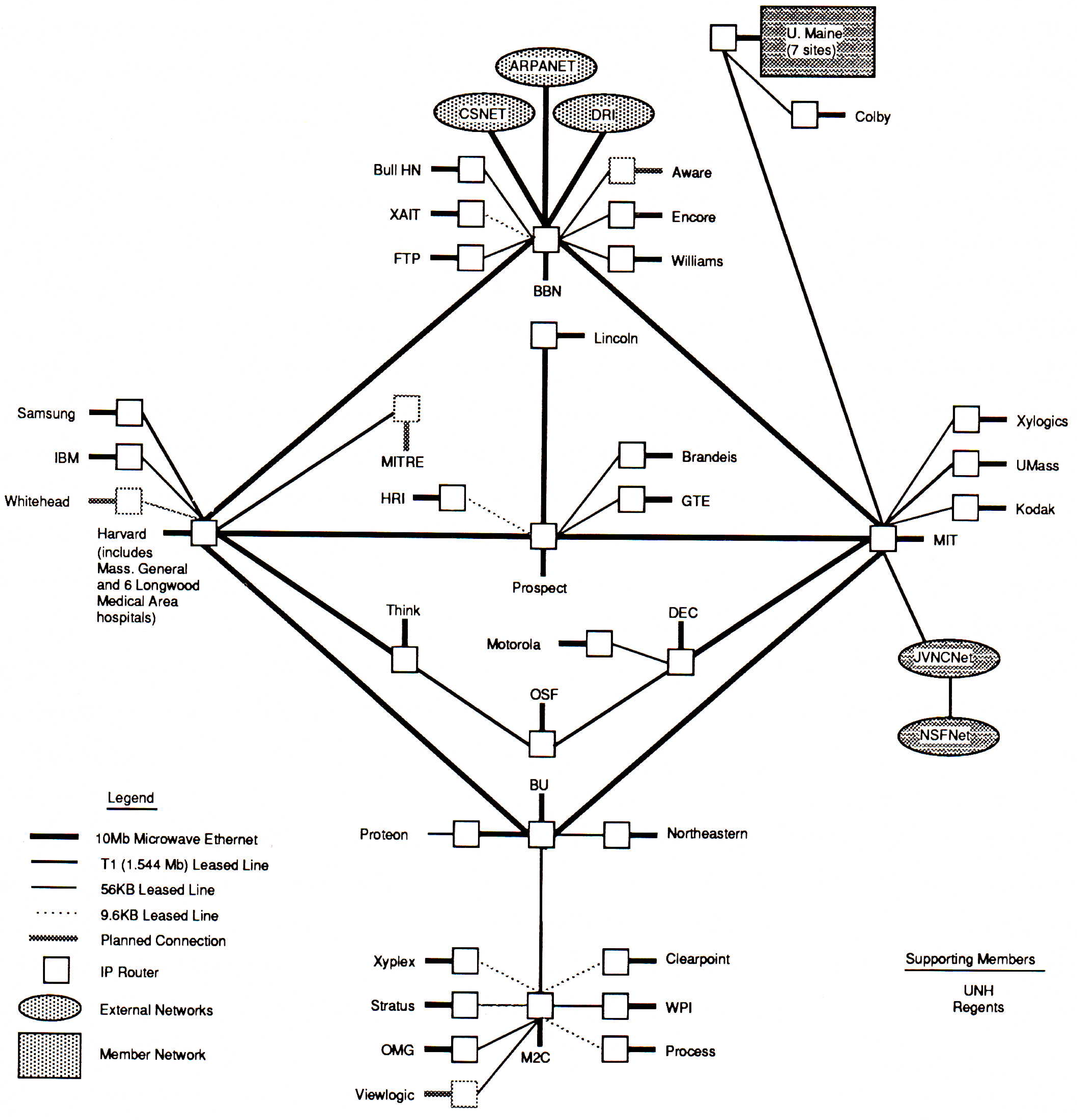

Figure 1: NEARnet Network Topology, March 1990.

NEARnet also provides online user services. There are several special interest electronic mailing lists that allow users to communicate with NEARnet staff and Planning Committee members. Documents of interest to members, including maps of the network, are available for anonymous FTP from `nic.near.net`. NEARnet also offers USENET news feeds to its members via NNTP.

continued on next page

Profile: NEARnet *(continued)*

Microwave technology

NEARnet serves the needs of both large and small sites by integrating several technologies that permit access at different speeds. These technologies include leased lines at speeds from 9.6Kb/s to T1, and 10Mb/s microwave links. NEARnet's topology consists of a series of interlocking rings. (See Figure 1.) Two main rings use 10-Mb/s microwave for transport; others use a mixture of various speed serial lines. By using microwave for its backbone, NEARnet is able to support full bandwidth 10-Mb/s Ethernet between members located miles apart. The 10-Mb/s capability offers users maximum flexibility for network services without compromising quality or response time.

NEARnet's microwave links consist of 23-GHz Motorola microwave radios and antennas, Microwave Bypass Systems Etherwave transceivers, and cisco Systems routers. The microwave links support standard full-bandwidth Ethernet transmission over standard microwave equipment through the use of an Etherwave transceiver. As configured in NEARnet, each microwave link is essentially a point-to-point full-duplex connection between pairs of cisco routers. (See Figure 2.) The cisco routers use cisco's IGRP to route traffic through the network and to redirect traffic around link failures.

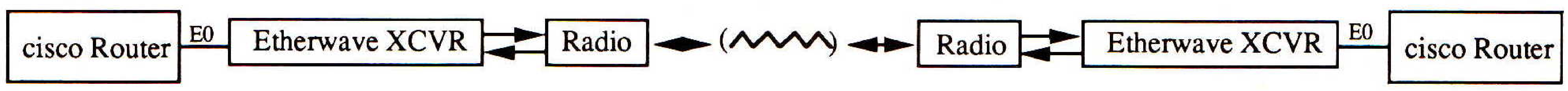

Figure 2: Standard Microwave Ethernet Link

To keep within the Ethernet's round-trip propagation delay budget of 46.4 microseconds, microwave path distances must be less than 4.3 miles. While the 4.3-mile distance limitation was no problem for most of NEARnet's 10-Mb/s members, one member site is twelve miles from the nearest backbone point of presence. NEARnet Technical Committee members developed a configuration to avoid collisions by splitting the full-duplex microwave path into two separate one-way, or simplex, Ethernets (a trick made possible by the Etherwave transceiver's design). A pair of unmodified Ethernet interfaces on the routers at each end of the link create a full-duplex path out of the two one-way paths. (See Figure 3.) This configuration avoids collisions because only one station transmits onto each Ethernet, allowing the microwave link to span distances up to the signal limit of about eight miles. However, routers typically don't understand one-way paths. cisco Systems modified their router software so that NEARnet could make the cisco routers understand the simplex paths and allow dynamic routing to operate over the dual simplex Ethernet configuration.

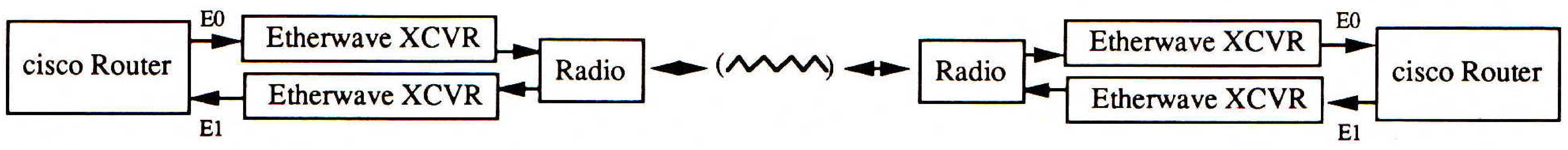

Figure 3: Extended Microwave Ethernet Link

Network analysis

NEARnet continues to explore ways to provide high-quality, cost-effective communications services to its members. NEARnet staff review performance of the network regularly using SNMP-based tools for gathering statistics on network throughput and link utilization. In these reviews, gateway and link reliability, MTTR, network throughput, and link utilization are examined. After each evaluation, NEARnet staff make recommendations such as upgrading links, changing operational procedures, etc.

Future directions

Ahead for NEARnet is improved high-speed connectivity to external networks, fiber-based enhancement of the backbone network for experimentation with high-speed technologies such as FDDI, and gateways to commercial services supporting research and development.

For more information about NEARnet, please contact John Rugo, NEARnet Accounts Manager, at 617-873-8730.

This article was written with the help of several people: Laura Breeden, manager of Network Services, BBN STC; James Bruce, vice president of Information Technology, MIT; Kent England, Network Systems Engineering Director, Boston University; Dan Long, network analyst, BBN STC; Cheryl Mammone, writer/editor, BBN STC; John Rugo, NEARnet project manager, BBN STC; and David Theodore, president of Microwave Bypass Systems.

Coming in next month's *ConneXions*

Routing protocols—what's all the fuss about? Who cares? Answer: if your organization is building or expanding an internetwork empire of host computers and/or LANs with multiple vendors, protocols, etc.—if you want to mix and match routers from different suppliers—*you* care. If you've got responsibility for DECnet and TCP/IP, and feel the winds of OSI blowing across your shoulders—if RIP won't cut it for you—if you're a vendor, network planner, or power user—*you* care.

What are the answers? What are the questions? Who are the players, what are they up to, and what will it mean to you? Why do vendors and planners all have different opinions—working from the same information base? And what's the IETF going to do about it?

Stay tuned for next month's "Route-Out at the OSI Corral: How Can You Be In Two Protocols At Once When You're Not OSPF At All, or, I've Got Algorithm." Starring Phill Gross, Ross Callon, Jack Haverty, Radia Perlman, Milo Medin, and a host of others. —*Daniel P. Dern*

Put your site on the map!

Doug Comer is working to update his book *Internetworking With TCP/IP—Principles, Protocols, and Architecture*. The revision will include two volumes (expected release late 1990). Along with this revision, he will update the *cover*, which you may recall is a map of the United States with points representing nodes on the Internet.

If you'd like your site to be represented on the cover of the second edition, send us electronic mail that specifies the geographic location of your site. Please use a geographic name that we can find in an atlas, instead of postal addresses that do not correspond to identifiable places. For example, "State College, PA" (the town) is better than "University Park, PA" (the university postal address).

Mail entries to: `tcpbook@purdue.edu`. If you are on the connected Internet (e.g., NSFnet), the subject line of your mail should be in the form: `Subject: YourCity, YourState, CONNECTED`. If your site uses TCP/IP but does not have full connectivity, the subject should be in the form: `Subject: YourCity, YourState, NOTCONNECTED`. If you are unsure about full connectivity, check with your local guru, or see if you can *ping* a machine on the Internet (e.g., try 128.10.2.1).

CONNEXIONS
480 San Antonio Road
Suite 100
Mountain View, CA 94040
415-941-3399
FAX: 415-949-1779

Bulk Rate
U.S. POSTAGE
PAID
SAN JOSE, CA
PERMIT NO. 1

CONNEXIONS

Subscribe to CONNEXIONS

U.S./Canada	$125. for 12 issues/year	$225. for 24 issues/two years	$300. for 36 issues/three years
International	$ 50. additional **per year**	**(Please apply to all of the above.)**	

Name ______________________ Title ______________________

Company ______________________

Address ______________________

City ______________________ State ____________ Zip ____________

Country ______________________ Telephone (   ) ____________

☐ Check enclosed (in U.S. dollars made payable to **CONNEXIONS**).

☐ Charge my ☐ Visa ☐ MasterCard ☐ Am Ex Card # ______________ Exp. Date ________

Signature ______________________

Please return this application with payment to: **CONNEXIONS**
480 San Antonio Road Suite 100
Mountain View, CA 94040
415-941-3399 FAX: 415-949-1779

Back issues available upon request $15./each
Volume discounts available upon request